AutoCV Challenge Design and Baseline Results

Zhengying Liu, Isabelle Guyon, Julio Junior, Meysam Madadi, Sergio Escalera, Adrien Pavao, Hugo Escalante, Wei-Wei Tu, Zhen Xu, Sebastien Treguer

AutoCV Challenge Design and Baseline Results

Liu Zhengying
U. Paris-Sud / U. Paris-Saclay
Inria Saclay
Orsay, France
zhengying.liu@inria.fr

Isabelle Guyon
U. Paris-Sud / U. Paris-Saclay
Inria Saclay
Orsay, France
guyon@clopinet.com

Julio C. S. Jacques Junior
U. Oberta de Catalunya / Computer Vision Center
Barcelona, Spain
jsilveira@uoc.edu

Meysam Madadi
Computer Vision Center
Barcelona, Spain
mmadadi@cvc.uab.es

Sergio Escalera
U. of Barcelona / Computer Vision Center
Barcelona, Spain
sergio@maia.ub.es

Adrien Pavao
Inria Saclay
Orsay, France
adrien.pavao@gmail.com

Hugo Jair Escalante
INAOE-CINVESTAV, Mexico
ChaLearn USA
hugo.jair@gmail.com

Wei-Wei Tu
4Paradigm Inc.
Beijing, China
tuwwcn@gmail.com

Zhen Xu
Ecole Polytechnique
Palaiseau, France
zhen.xu@polytechnique.edu

Sébastien Treguer
La Paillasse
Paris, France
streguer@gmail.com

Abstract

We present the design and beta tests of a new machine learning challenge called AutoCV (for Automated Computer Vision), which is the first event in a series of challenges we are planning on the theme of Automated Deep Learning[1]*. We target applications for which Deep Learning methods have had great success in the past few years, with the aim of pushing the state of the art in fully automated methods to design the architecture of neural networks and train them without any human intervention. The tasks are restricted to multi-label image classification problems, from domains including medical, areal, people, object, and handwriting imaging. Thus the type of images will vary a lot in scales, textures, and structure. Raw data are provided (no features extracted), but all datasets are formatted in a uniform tensor manner (although images may have fixed or variable sizes within a dataset). The participants's code will be blind tested on a challenge platform in a controlled manner, with restrictions on training and test time and memory limitations. The challenge is part of the official selection of IJCNN 2019.*

[1] https://autodl.chalearn.org/

1. Introduction

Machine learning, and deep learning in particular, has achieved considerable success in recent years. This in a wide variety of tasks ranging from computer vision, to natural language processing. State-of-the-art in deep learning, aligned with the increasing computational power of personal computers and accessible Graphic Processing Units (GPU) with strong computational capabilities, caused a revolution with respect to traditional computer vision, with unprecedented and outstanding results in different types of tasks such as image classification, recognition, and cap-

tioning, among others. However, so far this success crucially relies on human intervention in many steps (e.g., for data pre-processing, feature engineering, model selection, hyperparameter optimization, etc.). As the complexity of these tasks is often beyond non-experts, the rapid growth of deep learning applications has created a demand for off-the-shelf or reusable methods, which can be used easily and without expert knowledge. It is in this context that the Automated Deep Learning (AutoDL) field arises.

Previous and pioneer work on Automated Machine Learning (AutoML) in the context of supervised learning (see, e.g., [8]) comprises the basis for taking autonomy into the context of deep learning (AutoDL). In broad terms, the goal of AutoML is to develop "universal learning machines" capable of solving supervised learning problems without any user intervention. Great progress has been achieved in this topic, with quite competitive solutions (e.g., [5]) being publicly available to the average user. Such solutions, however, have been designed to work on tabular formatted data. That is, the feature extraction and representation strategy is responsibility of the user. AutoDL goes one step further, by attempting to automate the feature extraction and representation processes, in addition to the learning step. Ideally, with AutoDL we would like to find an autonomous solution that receives as input any type of input data and automatically generates a model that solves the associated task. As a first step in such direction, we organize the AutoCV challenge in which we ask participants to develop automatic methods for learning from raw visual information.

The objective of the new AutoCV challenge is to address some of the limitations of the previous challenges and to provide an ambitious benchmark for code wrappers around TensorFlow [1], which should be able to solve multi-label classification problems without any human intervention, in limited time, on any large-scale dataset. Data will be formatted in a uniform way as series of example-label pairs having identical corresponding structures, e.g. $\{X_1, Y_1\}$, $\{X_2, Y_2\}$, etc. This will lend itself in particular to the use of convolutional neural networks (CNN), where X_i are images, Y_i are binary vectors indicating which classes X_i belongs to. Although the use of Tensorflow will be encouraged by providing participants with a starting kit including sample code demonstrating how to solve the problems at hand, the participants will be free to provide solutions with any deep learning / machine learning framework, such as scikit learn [18] and PyTorch [17].

This paper describes the design of the AutoCV challenge, the first competition of its kind that aims at automating model construction for the analysis of visual information. A distinctive feature of this challenge is that models will be able to deal with raw data directly, hence making it a perfect testbed for representation learning and CNN based methods. Please note that although we expect deep learning models to obtain the best performance, participants can propose solutions based in any machine learning formulation. This competition is the first of a series that aims at boosting research on AutoDL.

The article is organized as follows. In Section 1.1, we provide a mathematical formulation of the central problem: the AutoML problem. In Section 1.2, we summarize related work. In Section 2, we present the overall competition design with details on the competition protocol, data, metrics used for evaluation, etc. In Section 3, we introduce several baseline methods and report their performance on formatted datasets. We conclude our paper in section 4.

1.1. Mathematical Formulation of the Problem

Since the AutoCV challenge aims to tackle the AutoML problem within the field of computer vision, we provide herein a mathematical formulation of the AutoML problem. We focus on the supervised learning case for current challenge.

Suppose we have a dataset $D = \{(x_i, y_i)\}_{i=1}^n$, associated to a supervidsed learning task, where $x_i \in \mathcal{X}$ are examples (e.g. images) and $y_i \in \mathcal{Y}$ are their corresponding labels (e.g. "dog" or "cat" or a binary vector). To define a *task*, we first do a *train/test split* $D = D_{tr} \cup D_{te}$ then separate examples and labels in test set to get $D_{te}^{\emptyset} = \{x | (x, y) \in D_{te}\}$ and $Y_{te} = \{y | (x, y) \in D_{te}\}$. A supervised learning task (either classification or regression) is then defined by the 5-tuple [2]

$$\mathcal{T} = (D_{tr}, D_{te}^{\emptyset}, L, B_T, B_S)$$

where $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a loss function measuring the performance $L(y', y)$ of predictions y' with respect to true labels y, B_T and B_S are time and space budget restrictions, respectively (these budgets are given with respect to a fixed universal Turing machine, e.g. a modern computer). The goal of the task is to find an *algorithm*

$$f : x \mapsto y$$

capable of mapping each example x to a prediction y of its corresponding label, within time and space budget limits B_T and B_S. Where we want this algorithm f to be as "good" as possible at making predictions. Such goodness is measured by a performance function $P(f; D_{te}, L)$ typically defined by

$$P(f; D_{te}, L) = -\frac{1}{|D_{te}|} \sum_{(x,y) \in D_{te}} L(f(x), y)$$

Then the problem can be formulated as (**Hard-coding Problem**)

$$\arg\max_f P(f; D_{te}, L). \tag{1}$$

[2] In some scenarios, the set of test examples $D_{te}^{\emptyset}$ is unknown at training time too and we have for theses cases $\mathcal{T} = (D_{tr}, L, B_T, B_S)$.

An important remark about (1) is that the test labels in D_{te} are *unknown* at training time. This also implies that the values of the objective function in (1) are unknown too. The final performance is only known at test time. Thus to find a satisfying solution for (1), some estimation techniques are often applied, using either human intuition or statistical methods (e.g. using a validation set).

To solve (1) automatically, machine learning approaches exploit the dataset D_{tr} in the task $\mathcal{T} = (D_{tr}, D_{te}^{\emptyset}, L, B_T, B_S)$. The key is to construct a *learning algorithm*

$$A : \mathcal{T} \mapsto f$$

that takes a task $\mathcal{T}$ as input and outputs an algorithm f, within time budget B_T and space budget B_S. The problem then becomes (**Machine Learning Problem**)

$$\arg\max_{A} P(\hat{f}; D_{te}, L), \text{ where } \hat{f} = A(\mathcal{T}). \tag{2}$$

Note that the maximization task of finding A in (2) is usually done by human experts, or more specifically by *machine learning experts*.

In real applications and machine learning research, the learning algorithm A is usually encoded by many hyperparameters (e.g. kernel of SVM, neural network architecture, optimizer parameters such as learning rate, regularizer parameters such as weight decay, etc), which turn out to be crucial to obtain acceptable performance. So to stress out the importance of these hyperparameters, sometimes we also write learning algorithms as

$$A_\lambda, \lambda \in \Lambda$$

with λ representing the hyperparameters of A_λ and Λ the hyperparameter space. Manually tuning these hyperparameters is a time consuming trial-and-error process, and this is where AutoML enters. A big part of AutoML's goal is indeed to automate the hyperparameter tuning process. This subdomain of AutoML is called *hyperparameter optimization* (HPO). An *HPO algorithm*

$$\mathfrak{A} : \mathcal{T} \mapsto A$$

aims to *automatically* solve (2), i.e. to automatically find a learning algorithm A given a task $\mathcal{T}$. A typical process of an HPO algorithm goes like: try different learning algorithms A_j (with different hyperparameters) on $\mathcal{T}$, estimate their performance P_j and in the end suggest the most promising one(s). In the literature, the HPO problem is sometimes defined as the Full Model Selection (FMS) problem [4] or the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem [22]. We recall the CASH problem as follows. Given training set D_{tr}, we do a k-fold cross-validation [12] by splitting D_{tr} into k equal-sized partitions $D_{tr} = D_{va}^{(1)} \cup \cdots \cup D_{va}^{(k)}$ and write $D_{tr}^{(i)} = D_{tr} \backslash D_{va}^{(i)}$. Consider a set of algorithms $\mathrm{A} = \{A^{(1)}, \ldots, A^{(l)}\}$ with associated hyperparameter spaces $\Lambda^{(1)}, \ldots, \Lambda^{(l)}$. The CASH problem is defined by

$$\arg\max_{A^{(j)} \in \mathrm{A}, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^{k} P(f_{ij}; D_{va}^{(i)}, L), \text{ where } f_{ij} = A_\lambda^{(j)}(D_{tr}^{(i)}).$$

We see that this CASH formulation actually tries to solve (2) automatically by using k-fold cross-validation as estimation of the test score $P(f; D_{te}, L)$. Again, as in the case of classic machine learning formulation, this CASH formulation for AutoML is but one possible solution to the real problem (2), with manual choice of A, $\Lambda^{(i)}$, cross validation as estimation, fold number k, etc.

In real life, machine learning experts never deal with only one single task $\mathcal{T}$ in their life. They gain experience by working on many different tasks $\{\mathcal{T}_j\}$ and often become more and more efficient at finding good learning algorithms. More formally, this process can be described as follows. The past experience is described by[3]

$$\mathfrak{D} = \{(\mathcal{T}_j, A_j, P_j)\}_{j=1}^{N}$$

with P_j being (an estimation of) the performance got by applying learning algorithm A_j to task $\mathcal{T}_j$. And we wish to construct a *meta-learning* algorithm

$$\mathcal{A} : \mathfrak{D} \mapsto \mathfrak{A}$$

that *learns* an HPO algorithm $\mathfrak{A}$ by exploiting past experience $\mathfrak{D}$. Note that the set of past experience $\mathfrak{D}$ can be dynamically enriched during the process of applying $\mathcal{A}$, since $\mathcal{A}$ can typically try a new learning algorithm A_{new} on a task $\mathcal{T}$ and get a performance P_{new}. Then we can append the 3-tuple $(\mathcal{T}, A_{new}, P_{new})$ to $\mathfrak{D}$ since it becomes past experience too. We note here the similarity between this process and what we have in *reinforcement learning*. Keeping the dynamic nature of $\mathfrak{D}$ in mind, we can consider hyperparameter optimization as a special case of meta-learning with $\mathfrak{D} = \emptyset$ at beginning and that the search of A is done only on a fixed task $\mathcal{T}$. Although in the literature, the term "meta-learning" lays more emphasis on the fact of using past experience on *different* datasets to a new dataset. So to avoid confusion, we'll call both an HPO algorithm or a meta-learning algorithm an AutoML algorithm.

But what is the objective of $\mathcal{A}$? Given a list $\mathfrak{D}_{te}$ of tasks to solve (recall that $\mathfrak{D}_{te}$ contains 5-tuples of the form $\mathcal{T} := (D_{tr}, D_{te}, L, B_T, B_S)$), one form of the objective can be formulated as (**AutoML Problem**)

$$\boxed{\begin{array}{c} \arg\max_{\mathcal{A}} \sum_{\mathcal{T} \in \mathfrak{D}_{te}} P(\hat{f}; D_{te}, L) \\ \text{s.t. } \hat{f} = \hat{A}(\mathcal{T}) \text{ and } \hat{A} = \hat{\mathfrak{A}}(\mathcal{T}), \hat{\mathfrak{A}} = \mathcal{A}(\mathfrak{D}). \end{array}} \tag{3}$$

[3] We can also consider $\mathfrak{D} = \{\mathcal{T}_j, A_j, f_j, P_j\}_{j=1}^{N}$ with trained $f_j = A_j(\mathcal{T}_j)$. Then we are talking about *transfer learning* instead of meta-learning. But here we'll make no distinction for simplicity.

From this definition, we see that an AutoML algorithm $\mathcal{A}$ should exploit $\mathfrak{D}$ (a dataset of tasks and past experience) and produce $\mathfrak{A}$ (an HPO algorithm), which will produce (or select) a learning algorithm $\hat{A}$ for each task $\mathcal{T}$, and the goal is to maximize the sum (or equivalently the average) of the performance on all tasks to solve in $\mathfrak{D}_{te}$. To add more clarity, we'll call $\hat{\mathfrak{A}} = \mathcal{A}(\mathfrak{D})$ the *meta-learning step*, $\hat{A} = \hat{\mathfrak{A}}(\mathcal{T})$ the *hyperparameter optimization step* and $\hat{f} = \hat{A}(\mathcal{T})$ the usual *supervised learning step*. And here we emphasize the importance of the imposed time and space budget limits B_T and B_S for each task. This restriction is crucial for an AutoML algorithm to be applicable in real-life scenarios.

In addition to the definition of the AutoML problem in (3), another important aspect that AutoML intends to explore is the *any-time learning* framework for AutoML algorithms. In an any-time learning setting, an AutoML algorithm $\mathcal{A}$ should produce (for each task $\mathcal{T}$) a ready-to-predict learning algorithm A_t (or simply f_t) at any timestamp $t \in \mathbb{R}_+$. And we judge the performance of $\mathcal{A}$ based on the sequence $\{A_t\}_{t\in\mathbb{R}_+}$ instead of considering just the final one A_T. This means that we want our AutoML algorithm to not only produce correct predictions at final time but also be fast and efficient. And this urges AutoML algorithms to carefully deal with exploration-exploitation trade-off. It's not hard to image that this any-time learning setting of the challenge will make participants' algorithms more robust and ready to be applied to real-life problem solving.

The problem approached in the AutoCV challenge is that of solving the AutoML problem defined in (3) for scene understanding tasks in an any-time learning setting.

1.2. State of the Art

According to Murdock et al. [16], while much manual engineering effort has been dedicated to the task of designing deep architectures that are able to effectively generalize from available training data, model selection is typically performed using subjective heuristics by experienced practitioners. Although freeing users from the troublesome handcrafted feature extraction by providing a uniform feature extraction classification framework, DCNNs still require a handcrafted design of their architectures [15]. Hence, most deep architectures for image classification learn shared image representations with a single model.

It is well known that lower layers of deep neural networks tend to represent low-level image features while higher layers encode higher-level concepts. Intuitively, and as reported in [16], categories that are more similar should share more information than those that are very different. Thus, training independent fine-grained models for these subsets of related labels could result in specialized features that are tuned to differentiating between subtle visual differences between similar categories.

Hierarchical deep networks [23] attempt to incorporate information from a known hierarchy to improve prediction performance without requiring architecture changes by learning multi-task models with shared lower layers and parallel, domain-specific higher layers. However, hyperparameters such as the location of branches and the relative allocation of nodes between them must still be specified prior to training. Ideally, model selection should be performed automatically, allowing the architecture to adapt to training data. As a step towards this goal, authors [16] proposed *Blockout*, an approach for simultaneously learning both the model architecture and parameters. Inspired by *Dropout* [10], Blockout can be viewed as a technique for stochastic regularization that adheres to hierarchically-structured model architectures.

Pérez-Rúa et al. [19] addressed the problem of finding an optimal neural architecture design for a given image classication task. They proposed the *Efficient Progressive Neural Architecture Search* (EPNAS) by aggregating two state-of-the-art in neural architecture search, i.e., *Sequential Model-Based Optimization* (SMBO) [13], and increasing training efficiency by sharing weights among sampled architectures [20]. Carreira-Perpinán and Idelbayev [3] addressed the task of pruning neural networks, which consists of removing weights without degrading its performance. In their work, pruning is formulated as an optimization problem of finding the weights that minimize the loss while satisfying a pruning cost condition. Nevertheless, the method neither has strong influence on the model architecture nor on the model selection. Ma and Xia [15] proposed a genetic DCNN designer to automatically generate the architecture of a DCNN for each given image classification problem. First, they partition a DCNN into multiple stacked meta convolutional blocks and fully connected blocks. Then, they use refined evolutionary operations, including selection, mutation and crossover to evolve a population of DCNN architectures. The main limitation of this approach, as mentioned by the authors, is that although the training of each generated DCNN is limited to 100 epochs, the proposed genetic DCNN designer has an extremely high computational and space complexity, due to storing and evaluating a large number of DCNN structures.

According to Cai et al. [2] techniques for automatically designing deep neural network architectures such as reinforcement learning based approaches have recently shown promising results. However, their success is based on vast computational resources, making them difficult to be widely used. A noticeable limitation is that they still design and train each network from scratch during the exploration of the architecture space, which is highly inefficient. In their work, a new framework toward efficient architecture search is proposed by exploring the architecture space based on the current network and reusing its weights. A reinforcement learning agent is employed as the meta-controller, whose

action is to grow the network depth or layer width with function-preserving transformations. Thus, previously validated networks can be reused, optimizing computational cost. The method was applied to explore the architecture space of the plain convolutional neural networks (no skip-connections, branching etc.).

As it can be seen, automated computer vision (i.e., model architecture design and selection, hyper-parameter setting, etc.) become an emerging research line, with a wide range of possibilities (i.e., with respect to applications and ways to be exploited), in particular when deep learning is employed. Different approaches have been proposed in the past few years, without a general solution. Furthermore, the difficulty of comparing existing approaches relies on the lack of benchmarks. This way, we expect the design of the present AutoCV Challenge can help to advance the state-of-the-art on this field.

2. Competition Design

This section describes the design of the AutoCV challenge. As previously mentioned, this competition asks participants to develop AutoML methods to solve scene understanding tasks and starting from raw data.

2.1. Competition Protocol

AutoCV challenge adopts the same competition protocol as the upcoming AutoDL challenge [14]. The evaluation process, for a single task, is shown in Figure 1. In this competition, the same submission will be evaluated on 5 different tasks and the evaluation is handled in parallel, in an asynchronous/parallel manner.

Some major differences between the AutoCV challenge and prior AutoML challenges [6, 7] are:

1. **Raw data:** Data are no longer pre-processed in a uniform feature vector representation; instead, they are formatted into TFRecords (a standard generic data format used by TensorFlow [1]) by keeping their raw nature. For datasets with images under standard compression formats (e.g. JPEG, BMP, GIF), we directly use the bytes as data and decode them on the fly to have a 3D tensor for each image.

2. **Large scale datasets:** For development, datasets will all be under 4GB (after compression), for practical reasons (See Section 2.2), however, for final testing, datasets of hundreds of thousands of examples will be used.

3. **Any-time learning:** The metric of evaluation (based on learning curves, see Section 2.3) will force the participants to provide algorithms that can make good predictions as early as possible.

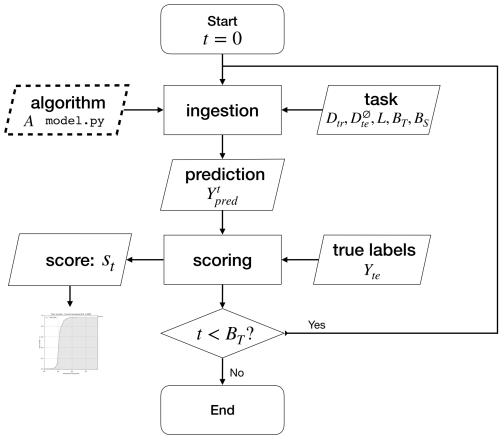


Figure 1: **AutoCV challenge's evaluation process** for *one* task defined by $D_{tr}, D_{te}^{\emptyset}, L, B_T, B_S$. Challenge participants have to provide a Python script named `model.py` implementing the logic of their AutoCV algorithm (shown as the dotted parallelogram). To evaluate its performance, this script is imported in ingestion program (defined in `ingestion.py`) in which it is trained on D_{tr} and produces a prediction Y_{pred}^{t} on $D_{te}^{\emptyset}$, where t is the timestamp. The prediction Y_{pred}^{t} is then compared to true labels Y_{te} in scoring program (defined in `score.py`) and produces a score s_t. This ingestion/scoring process is repeated until the time budget T is used up (or until `model.py` actively stops further training). At any time, the score sequence $s_{t_0}, s_{t_1}, ...$ is visualized as a learning curve and the area under learning curve is used as the evaluation for this task. Note that scoring is running in parallel with ingestion thus the time for computing scores is not counted in the time participants' code consumed (but the time used for training and predicting is).

4. **Fixed resource learning:** The participants will be given limited memory and computational resources to run their code. They will be informed of resources made available to them (number of cores, memory, GPU type, etc.). Their (compressed) code size will be limited to 300 MB, to allow submission of pre-trained models with moderate complexity. Thus, **pre-training** will be allowed, provided that the submission size limit is respected.

5. **Uniform tasks and metrics:** All problems will be multi-label classification problems and will be evaluated with the same metric (see Section 2.3).

One key aspect of this challenge, and other past AutoML challenges [8] we organized, is that it is a **code submission** challenge. Participants will submit code that will be trained

and tested on the challenge platform (see Sec. 2.5) without any human intervention, on datasets they will never see.

The challenge will be run in one single phase. Feedback score will be immediately provided on a leaderboard, using 5 **feedback datasets** (invisible to the participants, but visible to their submitted code). And this score is used for final evaluation.

In the challenge, we have 2 types of datasets: **public** and **feedback**. In contrast with some previous AutoML challenges [6, 7] in which the feedback data was distributed to the participants (except for the target values of the validation and test sets), in the AutoCV challenge, the practice datasets won't be exposed to the participants directly. Their code will be fully blind tested. However, several fully labeled "public" datasets will be provided, and we intend to set up a **public repository** to encourage participants to exchange among themselves other datasets and to enable meta-learning. A **starting kit** in Python with a TensorFlow interface will be provided (Sec. 2.4). Moreover, several examples of sample submissions will be provided.

2.2. Data

We provide 10 image datasets for the AutoCV challenge. They come from 5 different domains as follows: 1) **handwriting** recognition; 2) **object** recognition; 3) **People** related images (faces/emotions, etc); 4) **Aerial** images, and 5) **Medical** images. For either the 5 public datasets or the 5 feedback datasets, one dataset from each of the 5 domains is used. A detailed description about the 5 public datasets that are available to download is shown in Table 1.

All tasks are supervised learning problems, i.e., data samples are provided in pairs $\{X, Y\}$, X being an input image/tensor (which may have different number of channels for each domain) and Y a target binary vector. The problem is slightly simplified by the fact that for all images within a particular dataset, the bundle structure will be fixed.

2.3. Evaluation Metrics

AutoCV challenge enforces *any-time learning* by scoring participants with the Area under the Learning Curve (ALC) (Figure 2).

The participants can train in increments of a chosen duration (not necessarily fixed) to progressively improve performance, until the time limit is attained. Several predictions can be made during the learning process and this allows us to plot their learning curves, i.e., "performance" as a function of time. More precisely, for each prediction made at a timestamp, we compute for each (binary) class the *Area Under ROC Curve* (AUC), then normalize it (and average over all classes) by

$$NAUC = 2 * AUC - 1.$$

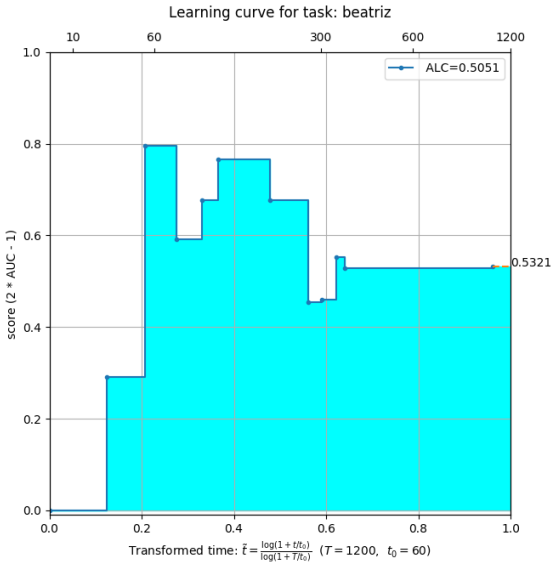


Figure 2: **Example of learning curve.** We modified the CodaLab competition platform (Sec. 2.5) so participants can save their results, at any intervals they choose, to incrementally improve their performance, until the time limit is attained. In this way, we can plot their learning curves: performance as a function of time. By evaluating them with the area under the learning curve, we push them to implement any-time learning methods. The x-axis corresponds to timestamp but normalized to [0,1]. This figure shows an example of possible over-fitting in which the participant could have stopped further training earlier.

Then for each dataset, we compute the Area under Learning Curve (ALC) of a submission as follows:

- at each timestamp t, we compute $s(t)$, the $NAUC$ (see above) of the most recent prediction. In this way, $s(t)$ is a step function with respect to timestamp t;
- in order to normalize time to the $[0, 1]$ interval, we perform a time transformation by

$$\tilde{t}(t) = \frac{\log(1 + t/t_0)}{\log(1 + T/t_0)}$$

  where T is the time budget (e.g. 1200 seconds = 20 minutes) and t_0 is a reference time amount (e.g. 60 seconds).

- then compute the area under learning curve using the

Table 1: **AutoCV datasets summary.** We provide 10 datasets for the AutoCV challenge, among which 5 **public** datasets are shown. Similar datasets have been formatted for the feedback and test phases, but will remain hidden for the participants. "row" and "col" are image size (height and width, respectively). "var" indicates the dimension is not fixed and can vary across examples.

#	Dataset	Domain	Class number	Sample number		Tensor dimension		
				train	test	row	col	channel
1	Munster	hand-writing	10	60000	10000	28	28	1
2	Chucky	objects	100	48061	11939	32	32	3
3	Pedro	people	26	80095	19905	var	var	3
4	Decal	aerial	11	634	166	var	var	3
5	Hammer	medical	7	8050	1965	600	450	3

formula

$$\begin{aligned} ALC &= \int_0^1 s(t)d\tilde{t}(t) \\ &= \int_0^T s(t)\tilde{t}'(t)dt \\ &= \frac{1}{\log(1+T/t_0)} \int_0^T \frac{s(t)}{t+t_0} dt \end{aligned}$$

we see that $s(t)$ is weighted by $1/(t+t_0)$, giving a stronger importance to predictions made at the beginning of th learning curve.

The ALC gives the evaluation score for one task. Finally, when ALC score is computed for all tasks, the final score is obtained by the **average rank** (over all tasks among all submissions). It should be emphasized that multi-class classification metrics are not being considered, i.e., each class is scored independently.

2.4. Starting Kit

To encourage participants to participate in the AutoCV Challenge, a starting kit, which can be downloaded from the competition web page, will be provided. The purpose of this starting kit is two-fold:

- **Submission file**: participants can directly upload this zip file as a submission. They can also deploy their own method and prepare a ZIP file for submission;
- **Local test environment**: it allows participants to have a complete local submission handling environment (for a single dataset). Thus they can easily run local tests, using their own method (implemented in model.py) for different dataset.

Both above utilities can be done following instructions in the `README.md` file, contained in the starting kit[4].

[4] https://github.com/zhengying-liu/autodl_starting_kit_stable

2.5. CodaLab: platform for running competitions

As in previous versions of AutoML Challenge, the AutoCV challenge will run on CodaLab. The CodaLab platform[5] is a powerful open source framework for running competitions that involve result or code submission. For AutoCV Challenge, we created our own CodaLab instance[6] so that computational resources can be adapted to the needs of the Challenge. CodaLab facilitates the organization of computational competitions from the organizers point of view, as well as provide a rich set of tools (e.g., forum, leaderboard, online submission, etc) to help participants. In AutoCV, participants will be able to submit their codes, and receive (almost) real-time feedback on a leaderboard (i.e., during a development phase), where the performances of different participants can be compared (as illustrated in Fig. 3).

3. Baseline Methods and Experimental Results

3.1. Baseline Methods

In this section, we present baseline methods with different model complexity and requiring different amount of computing resources, which have been applied on the datasets used in the AutoCV Challenge.

3.1.1 Linear Classifier with Basic Scheduling

This baseline method uses a very simple architecture: a neural network with no hidden layer. In the feed-forward phase, the input tensor is flattened then fully connected to the output layer, with a sigmoid activation. During training, it uses a sigmoid cross entropy loss (i.e., as if we are doing several binary logistic regression independently at the same time) and Adam optimizer [11] with default learning rate. The batch size is fixed to 30 for both training and test.

If the input shape is variable, some preprocessing procedure is required. When it is the case, we simply resize

[5] https://competitions.codalab.org/

[6] https://autodl.lri.fr

#	User	Entries	Date of Last Entry	<Rank> ▲	Dataset 1 ▲	Dataset 2 ▲	Dataset 3 ▲	Dataset 4 ▲	Dataset 5 ▲	Detailed Results
1	Zhengying	6	04/04/19	3.0000	0.0000(3)	0.0000(3)	0.0000(3)	0.0000(3)	0.0000(3)	Learning Curve
2	jullag	4	03/19/19	2.0000	0.1796(2)	0.0459(2)	0.0000(2)	0.0110(2)	0.0001(2)	Learning Curve
3	Pavao	9	03/17/19	1.0000	0.5839(1)	0.1899(1)	0.0007(1)	0.0386(1)	0.0002(1)	Learning Curve

Figure 3: Illustration of the leaderboard shown in our CodaLab instance, developed specifically for AutoCV.

all images to have fixed shape 112×112 (the number of channels is always fixed).

As we are in an any-time learning setting in AutoCV challenge, we need to have a working predictor at any time. According to the design of the challenge (Figure 1), this means that we also need a strategy for scheduling training and test. In this method, we propose following scheduling strategy. At the beginning, the algorithm trains the neural network for $s = 10$ steps. An estimation of time used per step is computed. Then the number of training steps gets doubled ($s \leftarrow 2s$) at each train/test call and the algorithm computes the duration required for this number of training steps using the estimation. This estimated duration is then compared to remaining time budget (sent by ingestion program). If there is still enough time, continue another call of training; otherwise, actively stop the evaluation process.

3.1.2 Convolutional Neural Networks Trained from Scratch

In this second baseline method, we use ResNet50 V2 [9]. The input image is resized to 128×128 and normalized with respect to the mean and standard deviation of all image pixel values. We assign sigmoid activation to the last layer for all types of classification tasks, i.e. multi-class and multi-label. The network is initialized with Xavier-Glorot initialization and trained with cross entropy loss and Adam optimizer with learning rate 0.001. We do not use any data augmentation during training. The data is shuffled in each epoch. The scheduling strategy is the same as the one defined in Sec. 3.1.1.

3.1.3 Neural Networks with Pre-trained Weights

In our third baseline method, we use pre-trained Inception V3 [21]. The Inception family appeals to many tricks in order to improve the image classification performance. All image inputs are resized to be 299×299. For grey images, we convert it to 3 channel and for images more than 3 channels, we extract the RGB channel. We use pre-trained weight provided by Tensorflow and fine-tune the model with cross entropy loss and default Adam optimizer. No data augmentation nor advanced techniques are used. Data is shuffled every epoch. The scheduling strategy is the same as that in section 3.1.1.

3.2. Experiments on Public and Feedback Datasets

We ran above baseline methods on the formatted datasets. All these experiments are carried out on Google Cloud virtual machine instances under Ubuntu 18.04, with one single GPU (Nvidia Tesla P100) and 16 GB Memory. The time budget is fixed to 20 minutes for all tasks.

The results are presented in Table 2. We show results on public and feedback datasets.

From Table 2, we see that the difficulty of different tasks vary a lot. Indeed, the difficulty depends on many parameters such as image shape, number of examples, number of classes, etc. This imposes participants' algorithm to be as flexible and robust as possible to provide satisfying any-time solution.

We note that in some cases, even though the final performance of ResNet/pre-trained Inception is better than that of linear baseline (e.g. in the case of Munster or Saturn), its overall ALC score is not better than linear baseline. This can be explained by the fact that the training is longer for ResNet (when the number of training steps is fixed) but the performance is not significantly better at the starting point. Thus it gains less Area under Learning Curve at beginning.

4. Conclusion and Further Work

We presented the design of a new challenge to stimulate the AutoML community to embrace deep learning and tackle the hard problems of automating architecture design and hyper-parameter search for models trained directly on raw data. We have run baseline methods and verified the feasibility of the tasks, given the allotted time and computational resources. The results will be reported at IJCNN 2019 and feedback from the community will be sought.

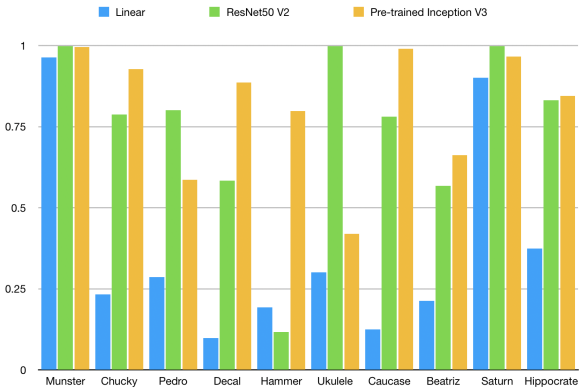
5. Acknowledgements

This project receives support from Google Zurich, ChaLearn and 4Paradigm Inc. This work has been partially supported by the Spanish project TIN2016-74946-P (MINECO/FEDER, UE) and CERCA Programme / Generalitat de Catalunya. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. This work is partially supported by ICREA under the ICREA Academia programme. We would like to thank our numerous collaborators for

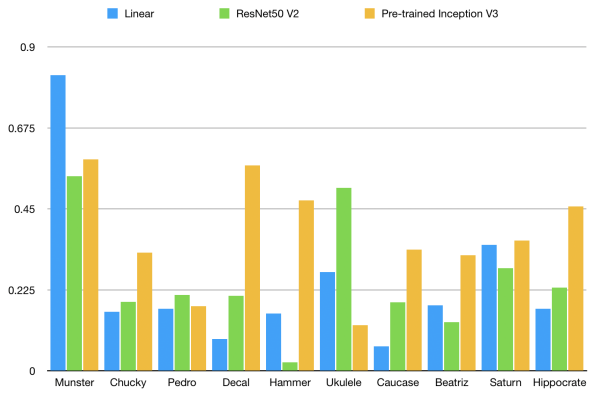
Table 2: **Baseline results** on public and feedback datasets used in AutoCV challenge. Three baseline methods (linear, ResNet and pre-trained Inception) are applied to all 10 datasets. Performances are in final normalized AUC (NAUC) and Area under Learning Curve (ALC). These results are further visualized in Figure 4a and Figure 4b.

	#	Dataset	Domain	Linear		ResNet50 V2 [9]		Pre-trained Inception V3 [21]	
				NAUC	ALC	NAUC	ALC	NAUC	ALC
Public	1	Munster	hand-writing	0.9628	0.8223	0.9999	0.5408	0.9950	0.5883
	2	Chucky	objects	0.2331	0.1643	0.7877	0.1914	0.9270	0.3289
	3	Pedro	people	0.2863	0.1733	0.8009	0.2115	0.5867	0.1805
	4	Decal	aerial	0.0982	0.0893	0.5833	0.2085	0.8861	0.5712
	5	Hammer	medical	0.1922	0.1596	0.1173	0.0238	0.7986	0.4742
Feedback	1	Ukulele	hand-writing	0.3003	0.2747	0.9986	0.5093	0.4189	0.1276
	2	Caucase	objects	0.1249	0.0683	0.7801	0.1910	0.9897	0.3367
	3	Beatriz	people	0.2129	0.1829	0.5675	0.1350	0.6621	0.3212
	4	Saturn	aerial	0.9003	0.3507	0.9987	0.2860	0.9665	0.3621
	5	Hippocrate	medical	0.3743	0.1726	0.8314	0.2319	0.8452	0.4571

Figure 4: **Visualization of baseline results.**



(a) Normalized AUC (NAUC) of the final predictions of the three baseline methods, after 20 minutes of training.



(b) Area under Learning Curve (ALC) of the three baseline methods, with 20 minutes of training.

their help and support. We thank André Elisseeff, Olivier Bousquet, Mahsa Behzadi, Tatiana Merkulova for helping challenge design and early development. Some of our datasets are collected and donated by Mehreen Saeed, Jun Wan, Stephane Ayache, Alexandre Gramfort, Hubert Jacob Banville, Sébastien Tréguer. We thank Lisheng Sun, Herilalaina Rakotoarison, Michèle Sebag, Marc Schoenauer, Kristin Bennett, Sébastien Tréguer, Danny Silver, Baiyu Chen, Shangeth Rajaa, Gavin Cawley, Albert Clapés i Sintes, Jun Wan, Quanming Yao, Mengshuo Wang, Yi-Qi Hu, Ju Xu, Jingsong Wang for participating in beta test and/or making useful suggestions. The CodaLab platform backend and web development are supported by Tyler Thomas and Eric Carmichael.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2, 5

[2] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient architecture search by network transformation. The Thirty-Second AAAI Conferenceon Artificial Intelligence, 2018. 4

[3] M. A. Carreira-Perpinán and Y. Idelbayev. learning-compression algorithms for neural net pruning. In Proceedings of the IEEE Conference on Computer Vision

and Pattern Recognition, pages 8532–8541, 2018. 4

[4] H. J. Escalante, M. Montes, and L. E. Sucar. Particle Swarm Model Selection. Journal of Machine Learning Research, 10(Feb):405–440, 2009. 3

[5] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, Advances in Neural Information Processing Systems 28, pages 2962–2970. Curran Associates, Inc., 2015. 2

[6] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. pages 1–8. IEEE, July 2015. 5, 6

[7] I. Guyon, I. Chaabane, H. J. Escalante, S. Escalera, D. Jajetic, J. R. Lloyd, N. Maci, B. Ray, L. Romaszko, M. Sebag, A. Statnikov, S. Treguer, and E. Viegas. A Brief Review of the ChaLearn AutoML Challenge: Any-time Any-dataset Learning Without Human Intervention. In Workshop on Automatic Machine Learning, pages 21–30, Dec. 2016. 5, 6

[8] I. Guyon, L. Sun-Hosoya, M. Boullé, H. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, et al. Analysis of the automl challenge series 2015-2018, 2017. 2, 5

[9] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In European conference on computer vision, pages 630–645. Springer, 2016. 8, 9

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580, 2012. 4

[11] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs], Dec. 2014. arXiv: 1412.6980. 7

[12] R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. event-place: Montreal, Quebec, Canada. 3

[13] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In The European Conference on Computer Vision (ECCV), September 2018. 4

[14] Z. Liu, O. Bousquet, A. Elisseeff, S. Escalera, I. Guyon, J. Jacques, A. Pavao, D. Silver, L. Sun-Hosoya, S. Treguer, W.-W. Tu, J. Wang, and Q. Yao. AutoDL Challenge Design and Beta Tests-Towards automatic deep learning. In MetaLearn workshop @ NIPS2018, Montreal, Canada, Dec. 2018. 5

[15] B. Ma and Y. Xia. Autonomous deep learning: A genetic DCNN designer for image classification. CoRR, abs/1807.00284, 2018. 4

[16] C. Murdock, Z. Li, H. Zhou, and T. Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2583–2591, June 2016. 4

[17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 2

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12(Oct):2825–2830, 2011. 2

[19] J.-M. Pérez-Rúa, M. Baccouche, and S. Pateux. Efficient progressive neural architecture search. arXiv preprint arXiv:1808.00391, 2018. 4

[20] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In J. Dy and A. Krause, editors, Proceedings of the 35th International Conference on Machine Learning, volume 80 of Proceedings of Machine Learning Research, pages 4095–4104, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. 4

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2818–2826, Las Vegas, NV, USA, June 2016. IEEE. 8, 9

[22] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. arXiv:1208.3719 [cs], Aug. 2012. arXiv: 1208.3719. 3

[23] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. Hd-cnn: Hierarchical deep convolutional neural networks for large scale visual recognition. In The IEEE International Conference on Computer Vision (ICCV), pages 2740–2748, December 2015. 4